Katrine Chow
CS 162 Winter 2018
Lab 3 Design and Reflection

Lab 3 – Design and Reflection Document

Problem:

Lab 3's goal is to implement a War game played by 2 players with dice. A unique feature of this assignment is that players have an option to select a loaded die, which:

1) Loaded Die: When rolled, on average rolls higher numbers than a regular die with the same number of sides.

Requirements:

a) A 2-player game.
b) Player who rolls the higher number scores 1 point. If both roll the same number, no points are awarded.
c) Die class represents a regular die. Player is able to specify the number of sides. This class has a member function that generates a random number from 1 to number of sides.
d) loadedDie class represents the loaded die. Player is also able to specify the number of sides. loadedDie class must inherit the behaviors of the Die class. A loaded die cannot roll a number higher than the number of sides it has.
e) Players can specify the number of rounds in the game.
f) Game class controls the entire flow of the game. It has a menu which allows players to start the game or exit. There is a function to ask users for all the relevant information and initializes member variables with that info. This class also stores and prints the game results after each round and at the end of the game.
g) Input validation for user inputs.
h) No memory leaks

Program Design:

See attached flowchart.

The program contains the Game class, the Die class, and the loadedDie class. A Menu class was re-used and tweaked from Project 1.

The Game class oversees the start to end of the game. It collects user information such as type of die, number of sides, and number of rounds, and has helper functions to store, get, and display results. The Game class also calls the Die and loadedDie class functions to simulate a dice rolls.

The Die class initializes N to the number of sides, and has a member function to generate a random number from 1 to N.

The loadedDie class also initializes its object with user inputs. It has its own randomizer function which overrides Die's own when loadedDie object is created. This function uses a different algorithm to rig the results.

Test Plan:

See attached PDF.

Reflection:

This lab is really fun to code and to play! I think I ended up spending an hour just playing the finished product because of the loaded die.

I originally expected to create a Die and loadedDie class, but was surprised to see the Game class requirement. I typically like to get user input and start a game from main, however I can see why this approach works better with the inheritance lesson. In a way, this design approach also helped me break down the program into smaller pieces for unit testing.

At first glance I didn't think I would need any pointers for this lab, but eventually I realized that I would need to use pointers to access the Die and loadedDie objects I've created within other functions.

I was also intimidated by coding most of the game mechanics within the Game class instead of main. However, this allowed me to practice keeping my main.cpp clean. I also had to design the program differently; sometimes I forgot that I don't need to pass anything or use a getter function within main.

I was pretty worried after Langton's Ant, but this lab gave me a chance to work on a better design. I was more careful about unit testing each feature before putting everything together. I also spent more time thinking about each class and functions. I scribbled several pages of notes before I began coding. This really helped me with keeping track of the program flow and with debugging.

I wanted to make the program look prettier, but I must move on to Lab 4 and Project 2. It is amazing how enjoyable it is to add the little touches. I'm glad I got to work on this lab. It is a confidence boost after the crazy first weeks.